Journal of Cybernetics and Informatics

published by

Slovak Society for Cybernetics and Informatics

Volume 5, 2005

http://www.sski.sk/casopis/index.php (home page)

ISSN: 1336-4774

Journal of Cybernetics and Informatics 5 (2005) 24-31
http://www.sski.sk/casopis/index.php

MODELING OF AGENT NEGOTIATION PROCESS

Oravec, V.*, Fogel J.**

**Institute of Informatics, SAS, Dúbravská 9, 84507 Bratislava, Slovakia*
***Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovičova 3, 81219 Bratislava*

upsyviki@savba.sk, jaroslav.fogel@stuba.sk

***Abstract**: Modeling of multi agent system is a very important part in the designing process. The paper presents an alternating-time temporal logic as such modeling approach. Alternating-time temporal logic (ATL) is temporal logic derived from branching temporal logic CTL. ATL models only behavior of multi-agent system on propositional level. Design of MAS can be more precise with alternating transition system (ATS), but more complicated. This paper tries to explain this complicating process by designing an illustrative multi-agent system example.*

Keywords: *multi-agent system, alternating-time temporal logic, alternating transition system, model checking*

1 INTRODUCTION

One of the best approaches how to model distributed computer systems is computation tree logic (CTL). This logic is used to model a closed system. It consists of two sets the first one being set of states, the other set of transitions between states. Thus, distributed system is represented with tree structure. Tree structure is composed of states as nodes and transitions as tree branches. Function of CTL is to describe every computation of distributed system in the tree structure. Each computation in a distributed system is represented by one path in the tree structure. CTL uses path quantifiers "A" and "E" and tense modalities "◊" and "□". Path quantifier "A" denotes that something is true for all paths. On the other hand, path quantifier "E" represents that something is true on some path. While path quantifiers concentrate on paths in tree structure, tense modalities concentrate on a particular path. Tense modality "◊" denotes that something is eventually true somewhere on the path. The next modality "□" expresses that something is always true on the whole path.

As mentioned before, this is very useful in distributed computer systems. Multi-agent system brings new view on distributed computer system [3]. Multi-agent system consists of several agents, but not all of them have to cooperate together, because they can create coalitions. Cooperation between two agents from different two coalitions is not desirable. This problem and more other problems are not solved by CTL. Multi-agent system has to be modeled by another logic. In this paper an alternating-time temporal logic [1] is presented, which can be used to model a multi-agent system. Alternating-time temporal logic describes multi-agent system behavior only. The structure is described by alternating transition system [1], which is also in the scope of this paper.

This paper is divided into several sections. It begins with introduction (this section) and finishes by conclusions, the (last section). Introduction is followed by the section with definition of alternating transition systems. Definitions of several types, such as synchronous and asynchronous system, are proposed in that section. After definition of alternating transition system, reader proceeds to the third section, where alternating-time temporal logic is discussed. An illustrative example for description of multi agent system is introduced in the following fourth section. Definition of alternating transition system is given in Section 5, and Section 6, the model checking with definition of alternating-time temporal logic formulas is introduced.

2 DEFINITION OF ATS

Alternating Transition System (ATS) [1] can be viewed as 5-tuple $S = (\Pi, \Sigma, Q, \pi, \delta)$, where Π is a set of propositions, Σ is a set of agents, Q is a set of all states. $\pi : Q \rightarrow 2^{\Pi}$ is mapping of each state to the set of propositions that are true in the state *q*, where *q* is part of *Q*. Mapping $\delta : Q \times \Sigma \rightarrow 2^{2^{Q}}$ maps each state and agent to nonempty set of possible choices. Precise mathematical definitions and examples of alternating transition system can be found in [1, page 5-9].

The next state of system S generally depends on the choices of all agents, because the system S proceeds in the next state q iff $q = \prod_{a \in \Sigma} Q_a$ where $Q_a \in \delta(q,a)$ is a choice made by agent $a \in \Sigma$. Note that intersection is singleton and that this is general definition of ATS, it can vary from type to type.

Before discussing particular types of ATS, some new terms are required. Consider two states $q, q' \in Q$ and agent $a \in \Sigma$. We say that q' is *a-successor* of q if $Q' \in \delta(q,a)$ and $q' \in Q'$, denoted as $q' = succ(q,a)$. We say that q' is *successor* of q if $q' = succ(q,a)$ for all agents $a \in \Sigma$. q' is successor of q iff whenever is the system S in state q and all agents can cooperate so that q' will be the next state of system S. A *computation* of S is infinite sequence $\lambda = q_0 q_1 q_2,...$ of states such that for $i \geq 0$ the state q_{i+1} is successor of q_i. Then *q*-computation is computation started with state q. Definite position in computation is denoted as $\lambda[i]$, prefix as $\lambda[0,i]$ and suffix as $\lambda[i,\infty]$.

2.1 Synchronous ATS

Turn-based synchronous ATS

We say that ATS is turn-based synchronous iff for every state of Q $!\exists$ agent $a \in \Sigma$ where $|\delta(q,a)| = 1$, $\delta(q,a) = \{q'\}$ and $\{q'\} \subseteq \delta(q,b)$ for $b \in \Sigma \setminus \{a\}$. So turn-based transition system can be viewed as 6-tuple $S = (\Pi, \Sigma, Q, \pi, \sigma, R)$, where $\sigma : Q \rightarrow \Sigma$ maps each state to the scheduled agent a and $R : Q \times Q$ is total transition relation (q' is a successor of q iff $R(q,q')$).

Lock-step synchronous ATS

Consider $S = (\Pi, \Sigma, Q, \pi, \delta)$. The S is lock-step synchronous ATS iff the following conditions are satisfied:

- $Q = \prod_{a \in \Sigma} Q_a$. Assuming global state $q = (q[a_1], q[a_2], ..., q[a_n])$ and $\Sigma = \{a_1, a_2, a_3, ..., a_n\}$ set of all agents in system, where $q[a]$ is a component of (global) q, which denotes (local) state of agent a.
- Each agent can determine its next local state dependent on the state of other agents but not dependent on choices made by them. Thus, function δ can by replaced by set of transition functions $\delta_a : Q \rightarrow 2^{Q_a}$.

2.2 Asynchronous ATS

Turn-based asynchronous ATS

In all types of turn-based systems only one agent decides the next state of a system. In synchronous system it was function σ, which determines an agent; in asynchronous system, scheduler was created. Scheduler is an agent that proceeds in every state. One agent is chosen by the scheduler to proceed with it (scheduler is replacement of function σ in synchronous systems). As in lock-step synchronous ATS we consider a δ as set of local transition function δ_a for $a \in \Sigma$.

ATS is turn-based asynchronous if there exists an agent $sch \in \Sigma$ called scheduler and for every agent $a \in \Sigma$ and every state $q \in Q$ exists a local transition function $\delta_a(q) : Q \rightarrow 2^{2^Q}$ such that the following four condition are satisfied:

- For all states $q \in Q$ and agent $a, b \in \Sigma \setminus \{sch\}$, if $a \neq b$ then $\delta_a(q) \cap \delta_b(q) \neq \{\}$. We say that agent $a \in \Sigma$ is enabled in state q if $\delta_a \neq \{\}$.
- For all states $q \in Q$, we have $\delta(q, sch) = \{\delta_a(q)\}$ the agent $a \in \Sigma \setminus \{sch\}$ is enabled in q.
- For all states $q \in Q$ and all agents $a \in \Sigma \setminus \{sch\}$ that are not enabled in q, we have $\delta(q,a) = \{Q\}$. That is, if the agent $a \in \Sigma \setminus \{sch\}$ is not enabled, it does not influence the successor state.
- For all states $q \in Q$ and all agents $a \in \Sigma \setminus \{sch\}$ that are enabled in q, assuming $\delta_a(q) = \{q_1, ..., q_k\}$, we have $\delta(q,a) = \bigcup_{i=1,...,k} (Q \setminus \delta_a(q)) \cup \{q_i\}$. If the agent a is enabled in state q, it chooses a successor state in $\delta_a(q)$ provided it is scheduled to proceed. If, however, a is not scheduled to proceed in q, then it does not influence the successor state, which must lie in $Q \setminus \delta_a$ because of the first condition.

2.3 Fair ATS

In previous section we have discussed a system without any qualification of possible states in one choice. In some cases we want to block, disable some states, namely q-computations. This can be done by the fairness condition.

Consider $S = (\Pi, \Sigma, Q, \pi, \delta)$. A fairness condition Γ is a set of fairness constraints for S, where fairness constraint is defined as $\gamma : Q \times \Sigma \to 2^{2^Q}$ such that $\gamma(q,a) \subseteq \delta(q,a)$ for $\forall q \in Q$ and $\forall a \in \Sigma$. Constraints can be in two states *a-enabled, a-taken*. The first one is *a*-enabled. The fairness constraint γ is *a-enabled* if $\gamma(q_i,a) \neq \{\}$. The second one is *a*-taken. The fairness constraint γ is *a-taken* if $Q' \in \gamma(q_i,a)$ and $q_{i+1} \in Q'$.

Two new terms can be defined characterizing computations regarding to fairness constraints, with respect to a set of agents $A \subseteq \Sigma$. The computation λ is *weakly* $\langle \gamma, A \rangle$*-fair* if for each agent $a \in A$, either there are infinitely many positions of λ where fairness constraint γ is not a-enabled, or there are infinitely many position of λ where fairness constraint γ is a-taken. The computation λ is *strongly* $\langle \gamma, A \rangle$*-fair* if for each agent $a \in A$, either there are finitely many positions of λ where fairness constraint γ is a-enabled, or there are infinitely many positions of λ where fairness constraint γ is a-taken.

Notes:

If fairness constraint is strong then it is weak. Note that for fairness condition Γ a computation λ is strongly/weakly $\langle \Gamma, A \rangle$-fair iff the computation λ is weakly $\langle \gamma, A \rangle$-fair for $\forall \gamma \in \Gamma$. Also note that each prefix computation of λ can be extended into strongly $\langle \Gamma, A \rangle$-fair computation. At the end, note that a computation λ is strongly/weakly $\langle \Gamma, A_1 \cup A_2 \rangle$-fair, for $A_1, A_2 \in \Sigma$, iff computation λ is strongly/weakly $\langle \Gamma, A_1 \rangle$-fair and $\langle \Gamma, A_2 \rangle$-fair.

The fairness condition is useful when some states are intended to be excluded from execution, e.g. infinite repeating of some computations' chunks.

3 ALTERNATING-TIME TEMPORAL LOGIC

Alternating–time temporal logic is used to describe behavior of multi-agent systems [1, page 12-17]. With ATL it is possible to define formulas with respect to set of proposition and set of agents. Set of propositions and set of agents are finite sets. These sets are the same as in ATS model. Each formula has to consist of one or more propositions with symbols of some logic operations "¬", " ∧", "∨". ATL formula also includes path quantifiers and temporal operators (tense modalities). The meaning of path quantifier is different than in CTL. Path quantifier describes set of agents, which cooperates to satisfy a particular part of ATL formula. ATL distinguishes two types of path quantifiers. The first one $\langle\langle A \rangle\rangle$, where A is a set of agents, denotes that agents in set A can cooperate to make something true. On the other hand, $[[A]]$ expresses that agents of set A cannot cooperate to make something false, they cannot avoid it. ATL defines three temporal operators: ○ ("next"), □ ("always"), U ("until"). Temporal operator "◊" can also be also found in formulas and means that something will happen in the future ($\Diamond \varphi \equiv true U \varphi$).

<u>Example:</u>

Consider a crossroad with traffic lights. Each arriving car has to satisfy several rules. When there is the red light cars cannot cross the crossroad. When there is the green light cars can continuously cross the crossroad. The last case is that with the orange light. With the red and orange lights the next color will be the green one, with only the orange light next color will be the red one. Thus four ATL formulas for the crossroad can be written:

$$\langle\langle\ \rangle\rangle \Box \left(\langle\langle ligths \rangle\rangle red \to \langle\langle car \rangle\rangle wait \right) \tag{1}$$

$$\langle\langle\ \rangle\rangle \Box \left(\langle\langle ligths \rangle\rangle green \to \langle\langle car \rangle\rangle go \right) \tag{2}$$

$$\langle\langle\ \rangle\rangle \Box \left(\langle\langle ligths \rangle\rangle red \wedge orange \to \langle\langle car \rangle\rangle \circ go \right) \tag{3}$$

$$\langle\langle\ \rangle\rangle \Box \left(\langle\langle ligths \rangle\rangle orange \wedge \neg red \to \langle\langle car \rangle\rangle \circ wait \right) \tag{4}$$

Note that these rules give no advice to cars how to behave in cases with orange light. They only say that in the next step cars can or cannot cross the crossing. Cars have to decide whether they crosses the crossroad.

4 A NEGOTIATION BETWEEN TWO AGENTS – ILLUSTRATIVE EXAMPLE

Consider a part of multi agent system, which consists of two types of agents PA and DA; for simplicity consider only one PA agent and several DA agents (Figure 1). PA is an abbreviation of a process agent that evaluates some operations. Values of some variables are needed for each operation. These values are retrieved from DA (Data Agent). Before value of variable can be retrieved PA looks for some DAs (this searching is not included in this paper).

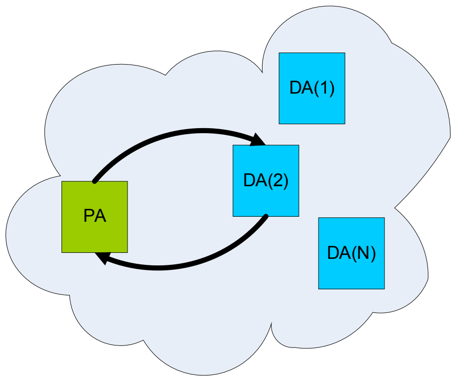

Figure 1. Considered part of a multi agent system

After particular DA is located, PA starts negotiations, i.e. queries whether DA supports certain variable. After the query has been sent by PA, it waits for replay. DA process evaluates this query. If DA does not support particular variable, DA will reply with REFUSE message (proposition VNA) and will finish the negotiation (proposition ER). Otherwise, if DA supports particular variable, it will reply with ACCEPT message (proposition VA) and wait for REQUEST message. PA's behavior depends on message received from DA. If REFUSE message is received (proposition VNA), then PA will finish negotiation (proposition ER). If ACCEPT message is received (proposition VA), then PA will continue the negotiation and send REQUEST message to DA. PA waits for DA's reply to REQUEST message. DA received REQUEST message and decides whether it is possible to send a value of queried variable. If it is possible (proposition VVA), then ACCEPT message will be sent by DA to PA and the negotiation will be finished (ALLOK). Otherwise, if it is not possible to send the value of variable (proposition VVNA), then REFUSE message will be sent by DA to PA and negotiation will be finished (proposition ER). After PA has received reply message from DA, PA decides about its next behavior. If ACCEPT message is received (proposition VVA), then PA will obtain the variable's value and finish negotiation (proposition ALLOK). On the other hand, if REFUSE message is received (proposition VVNA), then PA will know that something wrong happened and the behavior of PA will be as follows. PA can finish the negotiation or send another REQUEST message, and try to obtain the variable's value for the second time. PA decision whether to finish negation or send another REQUEST message depends on PA strategy. Note that one constraint for this strategy has to be set. Agent has to be able to finish the negotiation. For clarity, the whole negotiation process is shown in the next figure (Figure 2).

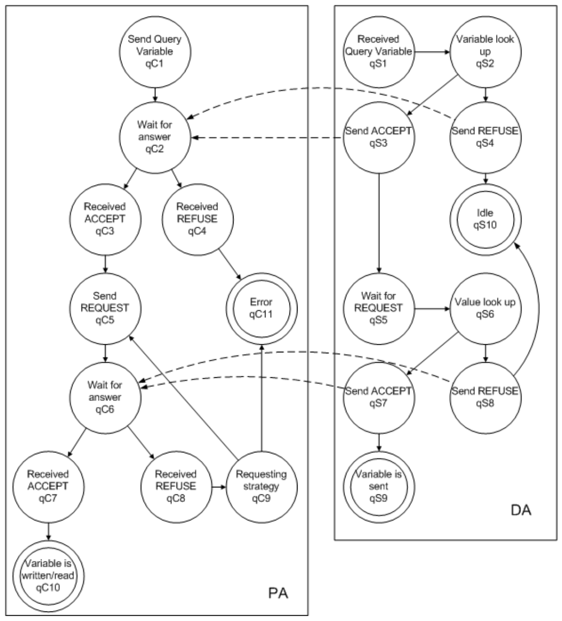

Figure 2. State chart for the negotiation process

5 DESIGN OF ATS

As mentioned in the second chapter, alternating transition system is described by 5-tuple $S = (\Pi, \Sigma, Q, \pi, \delta)$. In the previous chapter system S was described by words, but in this chapter we made the description according to this definition. Design of an ATS always begins with definition of three sets: set of agents Σ, set of propositions Π, set of states Q. There are only two agents in the whole system, thus

$$\Sigma = \{client, server\} . \tag{5}$$

Agent client tries to obtain the variable's value from agent server, which serves such queries. The set of propositions consists of six propositions: VA, VNA, VVA, VVNA, ALLOK, ER. The meaning of these propositions is given in Table 1.

proposition	*Description*
VA	Variable available
VNA	Variable not available
VVA	Variable's value available
VVNA	Variable's value not available
ALLOK	All right, variable's value is obtained
ER	Some error have occurred

Table 1 Explanation of propositions

Set of proposition is defined as follows:

$$\Pi = \{VA, VNA, VVA, VVNA, ALLOK, ER\} . \tag{6}$$

The last set, set of states Q, contains twenty-one elements.

$$Q = \{qC1, qC2, ..., qC10, qC11, qS1, qS2, ..., qS9, qS10\} \tag{7}$$

Ten of them are states of agent server, the remaining ones are the states of agent client. Each client's state name is qCx, each server's state name is qSx, where x is the number of particular agent's state.

Now we can proceed to definition of labeling and transition mappings. Labeling mapping π, as mentioned before, is used to map state of agent to propositions. Propositions are mapped to states, in which they are true. This mapping is useful in model checking. Behavior of ATS is compared with set of ATL formulas. Transition mapping describes the behavior of ATS precisely. It maps transitions between states. Our system is modeled by lock-step synchronous system, thus two transition mappings are defined - one transition mapping for each agent. Transition mapping δ_S is defined for agent server, transition mapping δ_C is defined for agent client.

Client		***Server***	
state	Propositions	state	propositions
qC1	-	qS1	-
qC2	-	qS2	-
qC3	VA	qS3	VA
qC4	VNA	qS4	VNA
qC5	VA	qS5	VA
qC6	VA	qS6	VA
qC7	VA,VVA	qS7	VA,VVA
qC8	VA,VVNA	qS8	VA,VVNA
qC9	VA,VVNA	qS9	VA,VVA,ALLOK
qC10	VA,VVA,ALLOK	qS10	ER
qC11	ER		

Table 2 Labeling mapping π

	qS1	qS2	qS3	qS4	qS5	qS6	qS7	qS8	qS9	qS10
qC1	qC2	qC2	qC2	qC2	qC2	qC2	qC2	qC2	qC2	qC2
qC2	qC2	qC2	qC3	qC4	qC2	qC2	qC2	qC2	qC2	qC2
qC3	qC5	qC5	qC5	qC5	qC5	qC5	qC5	qC5	qC5	qC5
qC4	qC11	qC11	qC11	qC11	qC11	qC11	qC11	qC11	qC11	qC11
qC5	qC6	qC6	qC6	qC6	qC6	qC6	qC6	qC6	qC6	qC6
qC6	qC6	qC6	qC6	qC6	qC6	qC6	qC7	qC8	qC6	qC6
qC7	qC10	qC10	qC10	qC10	qC10	qC10	qC10	qC10	qC10	qC10
qC8	qC9	qC9	qC9	qC9	qC9	qC9	qC9	qC9	qC9	qC9
qC9	qC5, qC11	qC5, qC11	qC5, qC11	qC5, qC11	qC5, qC11	qC5, qC11	qC5, qC11	qC5, qC11	qC5, qC11	qC5, qC11
qC10	qC10	qC10	qC10	qC10	qC10	qC10	qC10	qC10	qC10	qC10
qC11	qC11	qC11	qC11	qC11	qC11	qC11	qC11	qC11	qC11	qC11

Table 3 Transition mapping δ_C

	qC1	qC2	qC3	qC4	qC5	qC6	qC7	qC8	qC9	qC10	qC11
qS1	qS2	qS2	qS2	qS2	qS2	qS2	qS2	qS2	qS2	qS2	qS2
qS2	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4	qS3, qS4
qS3	qS5	qS5	qS5	qS5	qS5	qS5	qS5	qS5	qS5	qS5	qS5
qS4	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10
qS5	qS6	qS6	qS6	qS6	qS6	qS6	qS6	qS6	qS6	qS6	qS6

	qC1	qC2	qC3	qC4	qC5	qC6	qC7	qC8	qC9	qC10	qC11
qS6	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8	qS7, qS8
qS7	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9
qS8	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10
qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9	qS9
qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10	qS10

Table 4 Transition mapping δ_S

Implementation of decision into alternating transition system can be seen in transition functions. Agent server makes decision about variable's existence in state qS2 (Table 4). The next state is chosen from two possibilities (qS3, qS4). Decision in state qC2 of agent client (Table 3) is directly influenced by decision in state qS3 made by agent server. These states, so called decision nodes, are highlighted in Tables 3 and 4.

6 CHECKING OF DESIGNED MODEL

The crucial problem is to check the correctness of the designed system. This check is done by the model checking algorithm given in [1], which compares the designed system with the set of ATL formulas created from linguistic description of system. Such linguistic description was made in Section 4. Several ATL formulas can be defined. Some of them are presented here:

- Agents cannot avoid that their negotiation will finish with or without variable's value exchange.

$$[\![\ ]\!] \Diamond ((ALLOK) \vee (ER)) \tag{8}$$

- Decisions in agent *server* result in next step in agent *client.*
  - Whenever server supports queried variable, then client will know it in the next step.

$$\langle\langle\ \rangle\rangle \Box (\langle\langle server \rangle\rangle VA \rightarrow \langle\langle client \rangle\rangle \circ VA) \tag{9}$$

  - Whenever server does not support queried variable, then client will know it in the next step.

$$\langle\langle\ \rangle\rangle \Box (\langle\langle server \rangle\rangle VNA \rightarrow \langle\langle client \rangle\rangle \circ VNA) \tag{10}$$

  - Whenever server is able to sent value of queried variable, then client will know it in the next step.

$$\langle\langle\ \rangle\rangle \Box (\langle\langle server \rangle\rangle VVA \rightarrow \langle\langle client \rangle\rangle \circ VVA) \tag{11}$$

  - Whenever server is not able to sent value of queried variable, then client will know it in the next step.

$$\langle\langle\ \rangle\rangle \Box (\langle\langle server \rangle\rangle VVNA \rightarrow \langle\langle client \rangle\rangle \circ VVNA) \tag{12}$$

Several formulas were given above. It is simplier to create set of ATL formulas than alternating transition system for the same multi agent system. However from implementation point of view it is easier to implement alternating transition system then set of ATL formulas, because ATS is more detailed. It is obvious that both ATL formulas and ATS are needed in the design of multi-agent systems. Thus there is only one problem; to check that ATS was designed properly and satisfies all ATL formulas. The principle of model checking is very easy. All formulas have to be satisfied in all computations. This problem is EXPTIME-complete, so more efficient approaches have been invented. There exist model checking algorithms which are PTIME-complete.

The condition on behavior was set in the description of the multi-agent behavior (Section 4). Agent client has to have a requesting strategy (qC9) that agent is able to finish negotiation (qC10 or qC11). If this condition is not guaranteed, then the whole system can come to a deadlock. Simply, ATS system has to be extended with fairness condition Γ with strong fairness constraint γ.

$$\Gamma = \{\gamma\} \tag{13}$$

$$\gamma(qC9, client) = \{qC11\} \tag{14}$$

7 ACKNOWLEDGEMENT

The work presented in the paper was supported by following projects:

- APVT – 51 – 011602
- VEGA 2/4148/24

8 CONCLUSION

This paper concentrated on modeling multi-agent systems with alternating-time temporal logic and alternating transition systems. The negotiation between client and server was chosen as the application of such MAS design. As mentioned above, various principles how to model distributed computer systems can be used. We can use linear temporal logic, branching temporal logic, alternating-time temporal logic (ATL), etc. Mainly, the latter is the logic used in multi-agent system. It was invented for multi-agent system and is derived from branching temporal logic CTL. CTL describes the behavior of distributed computer system with states and transition between them into tree structure. ATL uses this principle, but changes it a bit. ATL is enriched with set of agents that are involved in computations described by formula.

As mentioned above ATL describes only behavior of multi-agent systems. Indeed, it describes structure of MAS, but not as precisely as ATS. Alternating transition systems are transition systems like final state machines with some advantages. These advantages help them create model of multi-agent systems. Three main types of ATS in previous sections have been introduced. Each of them has its own benefits. Turn-based synchronous system is very simple system. Its models are the most deterministic ones, and every single step in the system has to be synchronized. Another interesting type of ATS is lock-step synchronous system, where each agent is modeled by own transition mappings. It is obvious that every system's steps have to be synchronized (because of the synchronous type); but agents are modeled separately. The most general and natural system is the turn-based asynchronous system. Design of an asynchronous system is very simple, but in systems with huge number of states it is getting more complicated.

9 REFERENCES

[1] R. Alur, T.A. Henzinger, O. Kupferman. Alternating-time temporal logic. 38th IEEE Symposium on Foundations of Computer Science, pp. 100-109, 1997.

[2] W. van der Hoek and M. Wooldridge. Cooperation, Knowledge, and Time: Alternating-time Temporal Epistemic Logic and its Applications. In Studia Logica, 75(1):125-157, October 2003

[3] M. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. In Knowledge Engineering Review 10(2), 1995